REPORT DOCUMENTATION PAGE

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE 29 Apr 97	3. REPORT TYPE AND DATES COVERED

4. TITLE AND SUBTITLE
EFFICIENT IMPLEMENTATION OF IMAGE WARPING ON A MULTIMEDIA PROCESSOR

5. FUNDING NUMBERS

6. AUTHOR(S)
OWEN DANIEL EVANS

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
UNIVERSITY OF WASHINGTON

8. PERFORMING ORGANIZATION REPORT NUMBER
97-026

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
DEPARTMENT OF THE AIR FORCE
AFIT/CI
2950 P STREET
WRIGHT-PATTTERSON AFB OH 45433-7765

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION AVAILABILITY STATEMENT

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

19970505 133

14. SUBJECT TERMS

15. NUMBER OF PAGES
28

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT

DTIC QUALITY INSPECTED 1

Efficient Implementation of
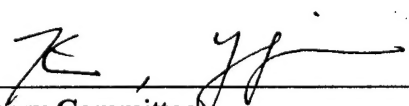# Image Warping on a Multimedia Processor

by

Owen Daniel Evans

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science in Electrical Engineering

University of Washington

1996

Approved by _____
(Chairperson of Supervisory Committee)

Program Authorized
to Offer Degree _____ *Electrical Engineering* _____

Date _____ *Dec. 13, 1996* _____

In presenting this thesis in partial fulfillment of the requirements for a Master's degree at the University of Washington, I agree that the library shall make its copies freely available for inspection. I further agree that extensive copying of the thesis is allowable only for scholarly purposes, consistent with "fair use" as prescribed in the U.S. Copyright Law. Any other reproduction for any purposes or by any means shall not be allowed without my written permission.

Signature _____

Date   13 Dec 96

University of Washington

Abstract

Efficient Implementation of
Image Warping on a Multimedia Processor

by Owen Daniel Evans

Chairperson of the Supervisory Committee: Professor Yongmin Kim
Department of Electrical Engineering

The spatial transformation of images, commonly known as image warping, is fundamental to many applications, e.g., remote sensing, medical imaging, computer vision, and computer graphics. Computational demands in image warping are high, requiring a geometric transformation, address and coefficient generation, and some form of interpolation. However, unlike most image processing algorithms, the data flow for image warping can be highly irregular, which makes any efficient implementation challenging.

This paper describes an efficient algorithm which addresses these challenges by making use of the capabilities of a single-chip multiprocessing microprocessor, the Texas Instruments TMS320C80 MVP (Multimedia Video Processor). The MVP's advanced digital signal processors (ADSPs) offer tremendous computational power through instruction-level parallelism and several key features designed for image processing. The MVP's intelligent input/output interface via the Transfer Controller (TC) permits efficient irregular memory accesses.

Affine and perspective warps have been implemented for 8-bit, 16-bit, and RGB color data using bilinear interpolation. The affine warp can generate 512 x 512 warped output images faster than real-time video rates require. For 8-bit images, the performance is 14.1 ms. Although the amount of computation necessary is the same for 16-bit images, the execution time increases to 15.2 ms since twice as many bytes need to be transferred. For RGB color images, it takes 28.0 ms. The perspective warp requires 46.3 ms for 8-bit and 16-bit images, and 60.4 ms for RGB color images. This unprecedented performance for software-based image warping exceeds many hardwired approaches reported in the literature.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

Image warping is a spatial transformation defining a geometric mapping between source and target images. A warp relates a source point p_s to a target point p_T by a transformation T, i.e., $p_T = T(p_s)$. The inverse transformation is specified as $p_s = T^{-1}(p_T)$. Figures 1 and 2 illustrate an example perspective warp using forward and inverse transformations, respectively. For the forward transformation, a grid is drawn in the source image. Depending on the perspective warp coefficients, the grid lines may converge in the target image. For the inverse transformation, a grid is drawn in the target image while the grid lines may converge in the source image. In Figures 1 and 2, the gray shapes designate the areas of valid image points. Points outside of this area are designated as background points. The source and target images consist of discrete points called pixels having intensity $I(p_s)$ and $I(p_T)$. A pixel's intensity representing its average brightness, also a discrete value, is limited by the image format. For example, the intensity for 8-bit images is limited to 2^8 or 0 to 255. The intensity of a target point $I(p_T)$ is generated by using a forward mapping with integration or an inverse mapping with interpolation. In general, since mapped points do not lie on discrete pixel locations, integration is used to construct the intensity of a target pixel from surrounding forward-mapped target points, and interpolation is used to reconstruct an inverse-mapped target point from surrounding source pixels. There are some advantages to an inverse-mapped approach. Integration requires computationally-expensive intersection tests to determine which points should contribute. Since the number of contributing points may vary, normalizing the integrated intensity is not a fixed or known element. The result must be checked if it exceeds the intensity range and clipped if necessary. In addition, holes or missing output points may occur if no surrounding points exist within a specified integration window. By using an inverse mapping, each target point is mapped uniquely, so no holes occur. In addition, a

fixed interpolation kernel is used requiring a fixed neighborhood of source pixels. For these reasons, an inverse mapping approach has been more widely used.

Image warping is fundamental to many applications, e.g., remote sensing, medical imaging, computer vision, and computer graphics [1]. For each of these application areas, image warping is often employed for different reasons. In remote sensing, the need is often on correcting geometric distortions occurred in image acquisition [2]. Typically, real images are not acquired from an ideal stationary pin-hole camera. In practice, the physical lens and the environment in which it operates can deform the acquired image, thus not faithfully representing the projection of points in the scene [2]. In medical imaging, we often need to integrate information from multiple imaging modalities (CT and MRI for anatomical information and PET or fMRI for functional information) or detect changes between acquired images [3]. A spatial transformation is used to register the images to aid in patient diagnosis and treatment planning [4]. Computer graphics applications involving image warping include texture mapping, ray tracing, and volume rendering.

With various applications generating an increasing number of images, there is a strong need for developing efficient image warping algorithms. For example, image registration uses many image warps in overlaying two images before detecting the optimum registration parameters. Depending on the search extent, this application may require hundreds of spatial transformations for each pair of images. To provide timely feedback to the clinicians and other specialists without using an expensive supercomputer, it is critical to use an efficient image warp. A machine vision application may require perspective view changes along with other image processing functions for each acquired frame. To meet timing constraints in numerous real-time machine vision applications, a fast image warp is essential.

2

Developing an efficient image warping algorithm is quite challenging considering the computational demands and data flow requirements. Computational demands are high, requiring a geometric transformation, address and coefficient generation, and some form of interpolation. An inverse transformation, defining a mapping between the output and input images, can involve simple multiplications and additions or costly divisions or transcendental functions. The mapped coordinate is used to generate the addresses of needed input pixels and interpolation coefficients. These input pixels are multiplied by the interpolation kernel coefficients and added together to generate the final target pixel value. In addition to these computational demands, an efficient data flow is critical to high performance. Working on adjacent target pixels may require irregularly-spaced source pixels. Irregular memory accesses must be supported without a severe penalty.

Recently, several commercially-available general-purpose processors have incorporated advances in computer architecture and VLSI technology to meet the demanding needs of digital video processing, HDTV, and networking environments [5]. Some of these new superscalar and very long instruction word (VLIW) processors employ instruction-level parallelism, which permits multiple operations to be executed in parallel per cycle. When combined with efficient parallel algorithms, these new processors can execute signal and image processing functions much faster than conventional processor architectures [6].

This paper presents a new image warping algorithm and its implementation for affine and perspective warps. The algorithm heavily utilizes the capabilities of the Texas Instruments TMS320C80 Multimedia Video Processor (MVP). A 14.1 ms execution time for an affine-warped 512x512 8-bit image using bilinear interpolation is better than any software-based algorithms and faster than even many hardwired approaches reported in the literature. Image warps have been implemented on numerous workstation platforms. Their execution times are typically 100 times or more slower than our performance. Parallel algorithms have been developed for several systems. In 1990, our group reported a performance of 2.2 seconds for a second-order warp on a 512x512 8-bit

3

image using bilinear interpolation using four floating-point processors [7]. More recently, another group at the University of Washington reported that the MasPar-1 parallel image processing computer with 16,384 processing elements requires 67.9 ms to complete a 512x512 affine warp using bilinear interpolation [8]. Researchers at the Princeton University developed a hardwired board and reported a performance of 25 ms to rotate, i.e. a special affine warp, a 512x512 8-bit image [9]. Siegel and Goetz-Greenwald [10] used the Datacube's Warper MKII and reported an execution time of 26 ms to warp a 512 x 512 image using a second-order mapping and 4x2 interpolation and 52 ms to warp a 512 x 512 image using a third-order mapping and 4x4 interpolation.

Chapter 2

Affine and Perspective Warping

An affine warp is very useful since it can handle several common geometric transformations such as scaling, shearing, rotation, translation and flipping. Figure 3 is an example affine warp in which the image has been rotated, stretched vertically, and translated. It has been also clipped to a finite output image window. An affine warp is specified by the following first-order analytic expressions:

$$x_S = a11\mathrm{x}_T + a21\mathrm{y}_T + a31 \qquad\qquad \text{Eq. (1)}$$

$$y_S = a12\mathrm{x}_T + a22\mathrm{y}_T + a32 \qquad\qquad \text{Eq. (2)}$$

where coefficients $a11$, $a21$, $a31$, $a12$, $a22$, and $a32$ are arbitrary real numbers. As shown in Figure 3, a first-order mapping produces the uniformly-spaced grid of points. Function $T^{-1}(p_T)$ generates a source point p_s for every target pixel p_T. Invalid points are also identified in Figure 3. This situation occurs when p_T generates p_s located outside of the source image boundaries. In case of an invalid point, the intensity of the target pixel $I(p_T)$ is set to a background pixel's value.

For each target pixel, p_s is used to generate the interpolation coefficients and the memory addresses of all needed source pixels. Interpolation is performed to resample the source image at a sub-pixel location. To determine the intensity of a target pixel, the source pixels surrounding p_s are multiplied by interpolation coefficients and their products accumulated. The interpolation coefficients determine how much of each source pixel would contribute to the final target pixel's intensity. The size of the interpolation kernel (thus, the number of needed source pixels) varies depending on the type of interpolation. We have used bilinear interpolation in our warping algorithms. As shown in Figure 4, bilinear interpolation with a kernel size of 2x2 uses the four surrounding pixels. Four coefficients which correspond to the pixels 1 through 4 in Figure 4 are:

$$\begin{bmatrix} c_1 & c_2 \\ c_3 & c_4 \end{bmatrix} = \begin{bmatrix} 1 - y_{S,frac} \\ y_{S,frac} \end{bmatrix} \begin{bmatrix} 1 - x_{S,frac} \\ x_{S,frac} \end{bmatrix}^{transpose}$$

Eq. (3)

These coefficients are a function of the fractional source point $p_{S,frac}$ (composed of $x_{S,frac}$ and $y_{S,frac}$) relative to the pixel 1.

To generate the memory addresses of the needed input pixels, the following equation is used.

$$Memory\ Address = Base\ Address + pitch \bullet y_{S,whole} + pixel\ size \bullet x_{S,whole}$$

Eq. (4)

Base address is the byte address of the upper-left corner pixel in the source image. *Pitch* is the memory offset between successive source image rows. *Pixel size* is the number of bytes per pixel. Using the whole number components of p_S, Eq. (4) generates the base pixel's memory address. Adjacent pixels can be easily located in reference to this memory address.

As already shown in Figures 1 and 2, a perspective warp alters the spatial relationship of pixels so that lines may converge to a vanishing point. The transformation is considered an image projection onto a new viewing plane created by a relative change in position of the image or the observer. Mathematically, the perspective warp is generated by dividing two affine warps point-by-point.

$$x_S = \frac{a11x_T + a21y_T + a31}{a13x_T + a23y_T + a32}$$

Eq. (5)

$$y_S = \frac{a12x_T + a22y_T + a32}{a13x_T + a23y_T + a33}$$

Eq. (6)

Compared to Eqs. (1) and (2), the common denominator in Eqs. (5) and (6) introduces the effect of converging lines. Interpolation and memory addressing for perspective warping work in the same way as with affine warping.

Chapter 3

Description of MVP Architecture and Key Features

The Texas Instruments TMS320C80 Multimedia Video Processor (MVP) is a single-chip multiprocessing microprocessor designed for high-speed image processing and real-time multimedia applications [11]. Figure 5 shows the block diagram of the MVP's internal architecture. It contains a RISC processor called the Master Processor (MP), four Advanced Digital Signal Processors (ADSPs), and a programmable Direct Memory Access (DMA) controller called the Transfer Controller (TC). The MP has a floating-point unit which can issue floating-point operations on every cycle. Each ADSP has a unique parallel architecture optimized for pixel operations. It contains a 16-bit fixed-point multiplier, a 3-input 32-bit ALU, and two load/store units, which can each be utilized concurrently. The TC allows various types of complex address calculations and data transfers.

MVP also has 50 kbytes of on-chip memory accessible in a single cycle. The memory is organized as 25 blocks of 2-kbyte modules, and each module is designated as an instruction cache, data cache, or data RAM. An instruction cache is assigned to the MP and each ADSP, but the data cache is available only to the MP. For the ADSPs, the data RAMs serve as the local storage area. While the cache memory is automatically serviced by the hardware to read from and write to the external memory, the data RAM needs explicit management and transfer requests by the programmer in software. Although this places additional demands on the programmer, tremendous performance gains are achieved through its effective use. The processors and on-chip memory modules are fully interconnected via the crossbar switch which operates at the instruction clock rate. In case of simultaneous accesses to the same location (contention), the crossbar resolves it through priority-based scheduling.

Several key features of the MVP summarized below contribute to efficient image warping, and they will be discussed more in detail in the following chapter.

- The TC, an on-chip DMA controller which can be programmed to meet the irregular data access requirements associated with image warping.
- Multiple banks of fast on-chip memory used in a ping-pong double-buffering mode enable the computations performed by the ADSPs and data input and output performed by the TC to go on simultaneously without interfering with each other.
- Instruction-level parallelism in developing a highly efficient core processing loop.
- Dedicated address unit adders used for secondary arithmetic operations.
- Three zero-overhead hardware loop controllers replacing multiple loop count and conditional branch instructions.

Chapter 4

Implementation on the Texas Instruments TMS320C80

Image warping involves four stages: geometric transformation, source pixel transfer, interpolation, and target pixel transfer. Due to the geometric transformation stage, adjacent target pixels often require source pixels in an irregular fashion. These source pixels can be transferred individually or collectively. Even though our algorithm uses a collective approach, it is worthwhile to review a system which transfers source pixels individually for comparison. Using a hardwired approach, the Datacube's Warper MKII, an optional component of the Datacube's MaxVideo system, divides the image warping task among two hardwired boards. The MKII's address unit inverse-maps target points to generate memory addresses for the needed source pixels and lookup table (LUT) addresses for the needed interpolation coefficients. The MKII's interpolator performs interpolation by multiplying the fetched source pixels with the respective interpolation coefficients and accumulating the results. Transferring source pixels individually has one key advantage. No descriptive parameters are needed. Since source pixels and interpolation coefficients are fetched to generate individual target pixels, they are always in a specific linear order. The MKII's interpolator can blindly use the fetched source pixels and interpolation coefficients in a predetermined fashion, i.e., multiply and accumulate.

To transfer source pixels collectively, the minimum and maximum mapped coordinates must be found to calculate the necessary data transfer parameters. Although transferring source pixels individually can avoid these calculations, a fast implementation demands the use of single-cycle memory to avoid page miss penalties associated with accessing more popular dynamic random access memory (DRAM) . The MKII uses an expensive high bandwidth single-cycle memory buffer for the source image and coefficient LUT so that any source pixel or interpolation coefficient can be quickly referenced.

Rather than using dedicated hardware, our algorithm is designed for a software solution using the Texas Instruments TMS320C80 Multimedia Video Processor (MVP). The irregular data flow requirement associated with image warping is a critical issue to be addressed in any software approach. Without relying on dedicated single-cycle memory, we generate pixels collectively to equally distribute the irregular memory access penalty across a group of target pixels. As illustrated in Figure 6, the target image is divided into a series of blocks. A fixed block size is used so that parameters such as target block addresses are easier to maintain. Depending on the image dimension and selected block size, target blocks located along the right and bottom edges may be smaller than this fixed size as shown in Figure 6. By processing pixels in blocks, the algorithm benefits from local coherency, i.e., a group of neighboring points in the target image tend to map to a group of closely-spaced points in the source image. For each group of closely-mapped points, the bounding block of source points, identified in Figure 6 with dashed lines, is transferred to the MVP's on-chip memory. Since a bounding block varies in size and location, parameters describing the data flow must be updated for every block of target pixels. The MVP's programmable memory interface provides this flexibility. As shown in Figure 6, three different cases may occur depending on the mapped location. First, no source pixels are needed if the bounding block is located outside of the source image. For this situation, the target pixels are set to the black background value, i.e., zero. Second, the bounding block may be located across an image border. In this case, the bounding block is clipped so that only valid source pixels are referenced and transferred to on-chip memory. Third, the bounding block may contain entirely valid source pixels.

Figure 7 shows a sequence of steps taken by our algorithm in determining the needed source pixel blocks. For each target block, the values $x_{S,min}$, $y_{S,min}$, $x_{S,max}$, and $y_{S,max}$ define the bounding source block. To determine these parameters directly, all of the mapped source points for a specific target block must be compared. For some transformations, these comparisons can be avoided or reduced due to the properties of the transformation. As

10

shown in Figure 8, since the affine warp is a linear transformation, $x_{S,min}$, $y_{S,min}$, $x_{S,max}$, and $y_{S,max}$ are always generated by mapping a specific target block vertex point and these associations are identical for each target block throughout the entire image. For the perspective warp, determining the bounding block is a little more difficult. Due to the divisions needed in perspective warping, the associations between $x_{S,min}$, $y_{S,min}$, $x_{S,max}$, $y_{S,max}$, and the target block vertex points vary depending on the target block location. Figure 9 is an example in which $x_{S,min}$ and $x_{S,max}$ are generated from different target block vertices within the same image warp.

Returning back to Figure 7, after the bounding block of source pixels is calculated, the block is adjusted for interpolation. To generate a target pixel, the source pixels surrounding the mapped source point p_s are needed. To ensure that the surrounding source pixels for every mapped source point including those located near the block boundary are contained in a bounding source block to be fetched, we need to increase the block size slightly depending on the interpolation kernel size. This block-based collective fetching of the source pixels smoothes the need for irregular data accesses across the group of target block pixels.

Since the needed source block may contain background pixels, more calculations are needed to determine the source pixel transfer. If the source block is located entirely outside of the source image dimensions, no pixel is needed. In this case, no target block pixel is inverse mapped and interpolation is skipped. The target block is filled with background pixels. If this condition is not true, then there is at least one valid source pixel within the source block. The source block is tested whether it is located along an image border. This test separates blocks which contain exclusively valid source pixels and blocks which contain some valid source pixels. If any of the test conditions is true as shown in figure 7, the source block is filled with background pixels and the parameters describing the needed source pixels are clipped. When the source pixels are transferred to the MVP's on-chip memory, they overwrite some of the background pixels. The

11

resulting on-chip memory block is the desired mixture of background and source pixels. As shown in Figure 7, if the source block contains entirely valid source pixels, the bounding block parameters do not have to be adjusted.

The steps shown in Figure 7 are rather complex, but it ensures that all the pixels needed for warping a particular target block are available in fast on-chip memory, which improves the performance of the core processing tight loop responsible for inverse mapping all target points and interpolating the results. Since all source pixels needed for interpolating the mapped points in a target block are available on-chip, no checking of the mapped points is needed. Source pixels are simply referenced from the generated addresses. By knowing *a priori* that all memory addresses would be valid, several tight loop instructions used in checking the generated addresses can be eliminated.

Multiple banks of fast on-chip memory in the MVP enable simultaneous processing and data transfer. While the ADSPs are tasked with processing source pixels and computing the parameters for the next needed source pixel transfer (the steps in Figure 7), the TC is programmed to transfer the previously processed target pixels off-chip and the source pixels to be processed next on-chip. By using these multiple memory banks in a ping-pong double-buffering mode, we can avoid memory contention and improve the performance by hiding the data transfer time behind the processing time via running the ADSPs and the TC concurrently.

Instruction-level parallelism of the MVP has been heavily utilized to develop a highly efficient core processing routine. In each ADSP, arithmetic and memory access operations for interpolation, address generation and memory reads/writes are often simultaneously executed. For example, an interpolation multiplication is executed at the same time as the previous multiplication result is accumulated and the next source pixel needed is fetched. By having multiple execution units available in each ADSP to be utilized simultaneously and multiple processors available in the MVP, developing an

efficient implementation on the MVP can involve quite a lengthy optimization process. To achieve this instruction-level parallelism, algorithm steps need to be reordered frequently. For example, some instructions can be executed ahead of time in parallel with previous instructions to be able to free up the busy execution units for the following instructions, thus reducing the overall number of cycles. In addition, the same operation can sometimes be accomplished with different resources. For example, extracting the lower half-word of a 32-bit register can be done with the multiplier, ALU, or global address unit. The 16-bit multiplier can multiply the register by 1. The ALU can perform a 32-bit logical AND operation with a mask to remove the upper 16-bits. The global address unit is capable of extracting a byte or a half-word from a 32-bit register.

The ADSP's address units are used to generate the memory addresses of source pixels. It is used to add the source base address in generating the final memory address as defined in Eq. (4). It is also used to update the address pointing to successive rows by adding or subtracting the source pitch to or from the on-chip memory address.

To improve the overall performance even further, we often utilize the dedicated address unit adders for performing additional arithmetic operations simultaneously. Since the affine warp is a linear transformation, adjacent mapped pixels differ only by a constant value. This arithmetic operation can be realized by using the ADSP's address units, i.e., using the computed address as a result rather than using it for memory access. Since the perspective warp is a division of two affine warps, the numerator and denominator can be each computed using the same approach. Since all mapped coordinates must be offset by the minimum source block x and y coordinates, the address unit adders are used again to accomplish other needed arithmetic operations.

The MVP's three zero-overhead hardware loop controllers are very useful in significantly reducing the image warp tight loop length by not requiring multiple loop count and conditional software branch instructions. Since a fixed block of target pixels is

13

processed, the number of loop iterations is known, i.e., the block width and height. While the inner loop processes horizontal pixels, the outer loop updates parameters for the next row.

Chapter 5

Results and Discussion

For an 8-bit affine warp using bilinear interpolation, each ADSP determines a target pixel value every 10 clock cycles. Since there are four ADSPs, the TMS320C80 generates a target pixel every 2.5 clock cycles. For a 512x512 target image, the tight loop for the 8-bit affine warping should be completed in 13.1 ms with the TMS320C80 running at 50 MHz. The measured performance of 14.1 ms was obtained including the additional setup instructions needed per target block and other overhead. The measured execution time for a 16-bit affine warp was 15.2 ms. Since the number of computing cycles for the 16-bit warp tight loop is the same as that for the 8-bit warp tight loop, this result shows that 1.1 ms of additional I/O time is needed to transfer twice as many bytes per pixel. For a RGB color affine warp using bilinear interpolation, each ADSP determines a target pixel value every 19 clock cycles, resulting in 24.9 ms for a 512x512 target image. The measured execution time of 28.0 ms indicates that the I/O time exceeds the computation time. Since there are three 8-bit bands for color data, target pixels are generated using smaller image blocks than before due to the limited on-chip memory size. By processing smaller target blocks, there is a little more overhead compared to 8-bit and 16-bit affine warps. Three affine warp examples with various rotation and shearing angles, scaling factors, and translation amounts are shown in Figure 10.

For the perspective warp using bilinear interpolation, each ADSP determines an output pixel value every 33 clock cycles. Compared to the affine warp, the additional cycles are directly attributed to the divisions needed for a perspective warping geometric transformation. For a 512x512 target image, perspective warping should be completed in 43.3 ms with the TMS320C80 running at 50 MHz. The measured performance of 46.3 ms is close to the expected 43.3 ms performance indicating that it is a compute-bound operation due to the overhead in handling the next source block transfer.

Execution times for 8-bit, 16-bit, and RGB color data are summarized in Table 2. Two example perspective warps are shown in Figure 11.

The performance of our image warping algorithm can be compared to the performance that has been reported in the literature. Image warps have been implemented in a programmable fashion on numerous workstation platforms. Typically, execution times are at least 100 times or more slower than those reported in Tables 1 and 2. Parallel algorithms have been developed for several systems, but their performance is still much less than that of our algorithm. In 1990, our group reported a performance of 2.2 seconds for a second-order warp on a 512x512 8-bit image using bilinear interpolation using four floating-point processors [7]. More recently, another group at the University of Washington reported that image warping was implemented on the MasPar-1 parallel computer with 16,384 processing nodes [8]. The MasPar-1 took 67.9 ms to warp a 512x512 image using bilinear interpolation compared to 14.1 ms in the MVP .

With the advancement of VLSI technology, several researchers and companies have developed dedicated hardware systems for image warping. Researchers at the Princeton University have designed and implemented a board which claimed to achieve real-time video-rate image rotation [9]. Image rotation can be easily achieved via an affine warp as shown in Figure 10. They reported an execution time of 25 ms to rotate an 8-bit 512x512 image. They did not specify the type of interpolation used. The Datacube's Warper MKII takes 26 ms to warp a 512x512 image using 4x2 interpolation and 52 ms using 4x4 interpolation [10]. Since the Datacube's MKII can use a second-order transformation and 4x2 interpolation or a third-order transformation and 4x4 interpolation, a direct comparison to our results cannot be made. However, the execution times for our affine and perspective warps provides a basis for estimating performance with other geometric transformations. Since divisions are very computationally expensive, it is reasonable to assume that second-order and third-order warps can be implemented between the performances of the affine and perspective warps. For bilinear interpolation, our

performances of 14.1 ms for an affine warp and 46.3 ms for a perspective warp indicate that our implementation is close to the Datacube's hardwired approach.

Demonstrating the flexibility of a programmable approach, our algorithm supports multiple data formats. 8-bit, 16-bit, 32-bit CCCX (C represents an 8-bit color band and X represents an 8-bit don't care band), and 32-bit XCCC formats have been implemented. Since the MVP has a 16-bit multiplier, no additional computations are needed in handling 16-bit data. To transfer twice as much data, however, it takes slightly longer than the 8-bit affine warp. Support for multiple data formats may be quite challenging or impossible for a hardwired approach with limited programmability. For example, the MKII can operate on 16-bit data, but it takes twice as long since it must operate on the least significant byte and most significant byte separately. To warp color images, the MKII must operate on each band independently. Target points are inverse-mapped for each color band, instead of mapping once and interpolating all of the color bands.

Our affine and perspective warps have already been used as building blocks for several higher-level image processing routines. They are part of the University of Washington Image Computing Library developed for use in MVP-based systems to provide a portable infrastructure of low-level routines so that higher-level algorithms and applications can be quickly developed [12]. The affine warp has been heavily utilized in wavelet-based multiresolution image registration [13]. Since the reference image must be rotated many hundred times once for each angle of interest at each resolution level, it is critical to use an efficient image warp in order to provide timely feedback to the clinicians and other specialists. The perspective warp has been used in our visualization algorithm to render 3D ultrasound images [14]. Siemens Medical Systems Ultrasound Group has been using the affine warp as part of a new acquisition and display process called SieScape™, which is used to produce panoramic Extended Field of View (XFOV) images. Narrow fields of view acquired by ultrasound scanners can be placed side-by-side via image correlation to

provide a bigger picture. To align these multiple images within real-time video-rate constraints, our affine warp is heavily used.

Chapter 6
Conclusion

We have developed a very efficient warping algorithm and implemented both affine and perspective warping functions in software on a modern microprocessor. This algorithm makes use of a commercially-available programmable multiprocessing microprocessor incorporating advances in computer architecture and VLSI technology. Its performance exceeds any software-based approach reported in the literature. It also compares favorably with several hardwired implementations such as the Datacube's Warper MKII. Its demonstrated flexibility, support of multiple data formats and interpolation types, and performance indicate that a software-based solution can now meet the demands of many image processing applications without the use of dedicated hardware.

References

[1] Wolberg G., Digital Image Warping. IEEE Computer Society Press, Los Alamitos, CA, 1990.

[2] Breen L. and Bryant J., "Image warping by scanline operations," Comput. & Graphics. vol. 17, no. 2, pp. 127-130, 1993.

[3] Weinhaus F. and Walterman M., "A flexible approach to image warping," SPIE Proceedings. vol. 1244, pp. 108-122, 1990.

[4] Goshtasby A., "Piecewise linear mapping functions for image registration," Pattern Recognition. vol. 19, pp. 459-466, 1986.

[5] Pirsch P., Demassieux N., and Gehrke W., "VLSI architectures for video compression -- a survey," Proceedings of the IEEE vol. 83, pp. 220-245, 1995

[6] Basoglu C., Lee W., and Kim Y., "An efficient FFT algorithm for superscalar and VLIW processor architectures," Real-Time Imaging, sumitted, Feb. 1996.

[7] Wong, G., "The design and implementation of parallel algorithms for UWGSP3, a high performance image processing and graphics subsystem for the NeXT computer," MSEE Thesis, University of Washington, 1990.

[8] Wittenbrink C. M. and Somani A. K., "2D and 3D optimal parallel image warping," Journal of Parallel and Distributed Computing vol. 25, pp. 197-208, 1995.

[9] Ghosh I. and Majumdar B., "VLSI implementation of an efficient ASIC architecture for real-time rotation of digital images", International Journal of Pattern Recognition and Artificial Intelligence. vol. 9, pp.449-462, 1995.

[10] Siegel S. and Goetz-Greenwald B., "VME boards perform high speed spatial warping", SPIE Proceedings. vol. 1027, pp. 77-80, 1989.

[11] Guttag K., Gove R. J., and Van-Aken J. R., "A single chip multiprocessor for multimedia: The MVP," IEEE Comput. Graphics. Appl., vol. 12, no. 6, pp. 53-64, 1992.

[12] Kim J. and Kim Y., "UWICL: A multi-layered parallel image computing library for single-chip multiprocessor-based time-critical systems," Real-Time Imaging, vol. 2, pp. 187-199, 1996.

[13] Wu H., "Fast wavelet-based multiresolution image registration on a multiprocessing digital signal processor," MSEE Thesis, University of Washington, 1996.

[14] Deforge C., "Near real-time 3-D ultrasound: a feasibility study," MSEE Thesis, University of Washington, 1996.

<div align="center">Execution Time (ms)</div>

Affine warps	8-bit	16-bit	BGRX 32-bit
2x2 interpolation (bilinear)	14.1 ms	15.2 ms	28.0 ms

<div align="center">Table 1: Execution times for affine warps.</div>

<div align="center">Execution Time (ms)</div>

Perspective warps	8-bit	16-bit	BGRX 32-bit
2x2 interpolation (bilinear)	46.3 ms	46.3 ms	60.4 ms

<div align="center">Table 2: Execution times for perspective warps.</div>

Source Image

Target Image
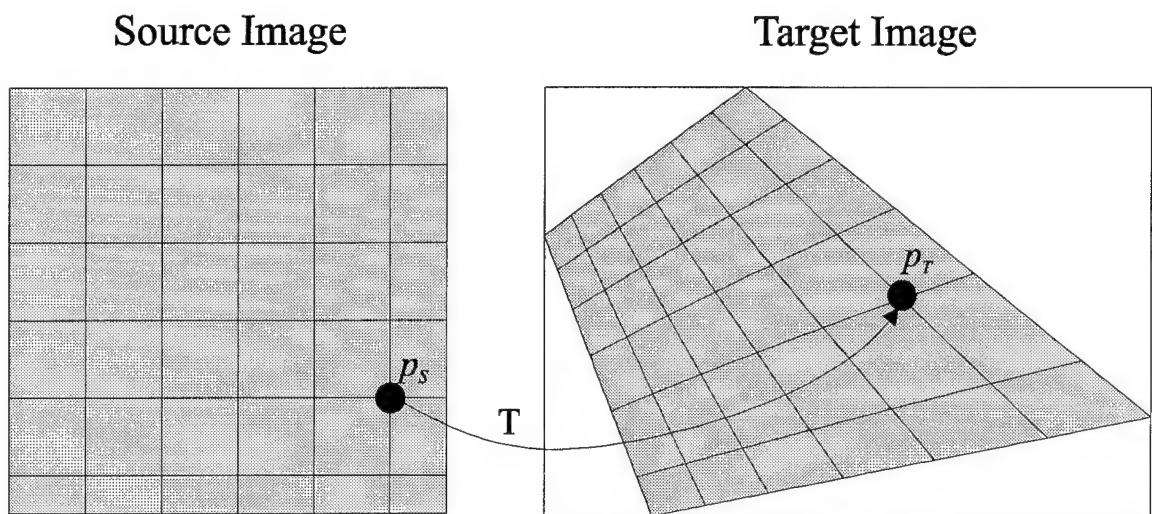
Figure 1: Forward transformation of a perspective warp.

Source Image

Target Image
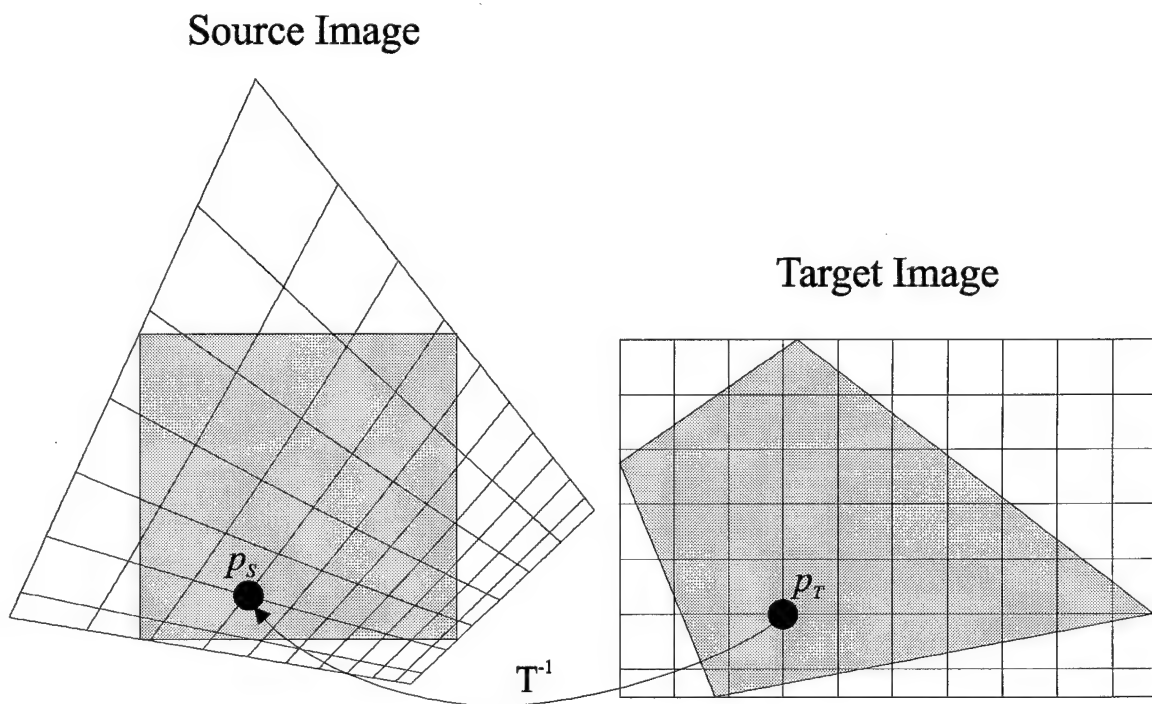
Figure 2: Inverse transformation of a perspective warp.
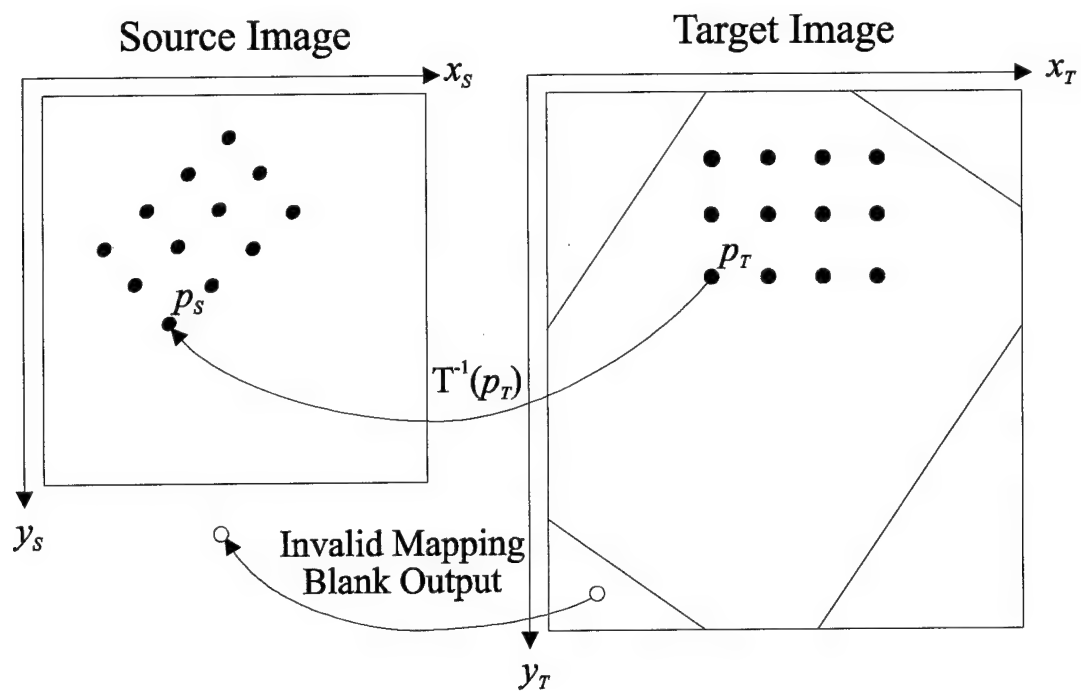
Figure 3: Affine warp.
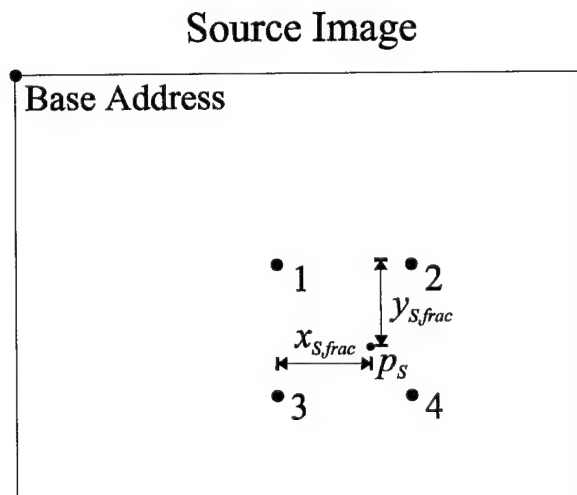
Figure 4: Bilinear interpolation.
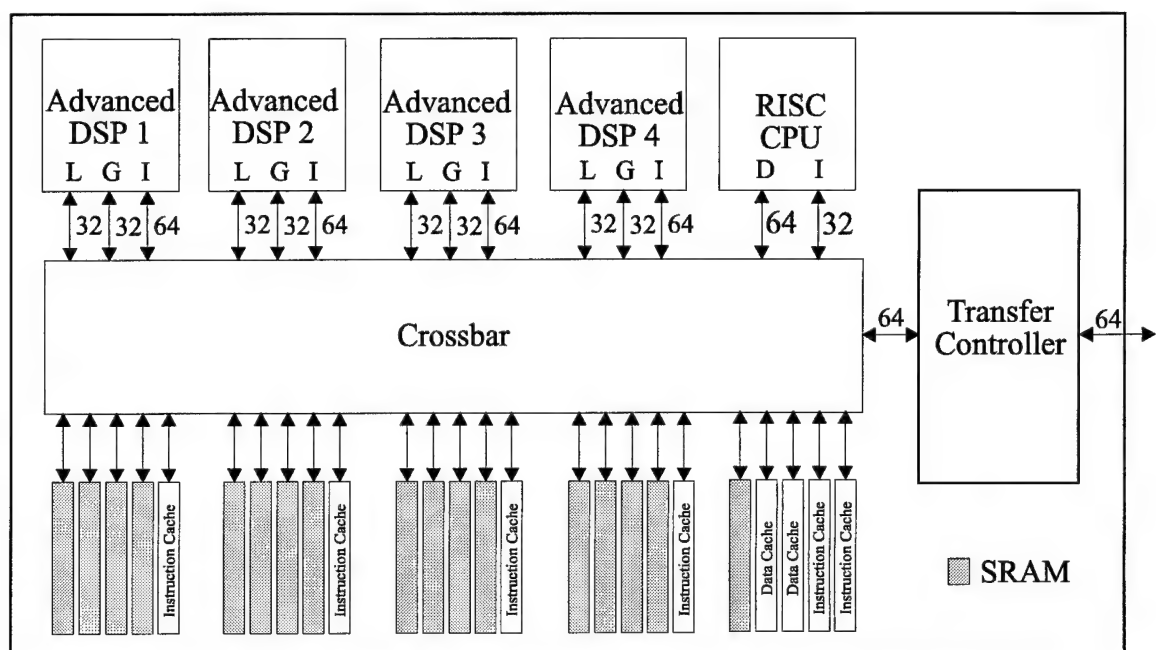
Figure 5: Internal architecture of the TMS320C80.

Source Image

Target Image
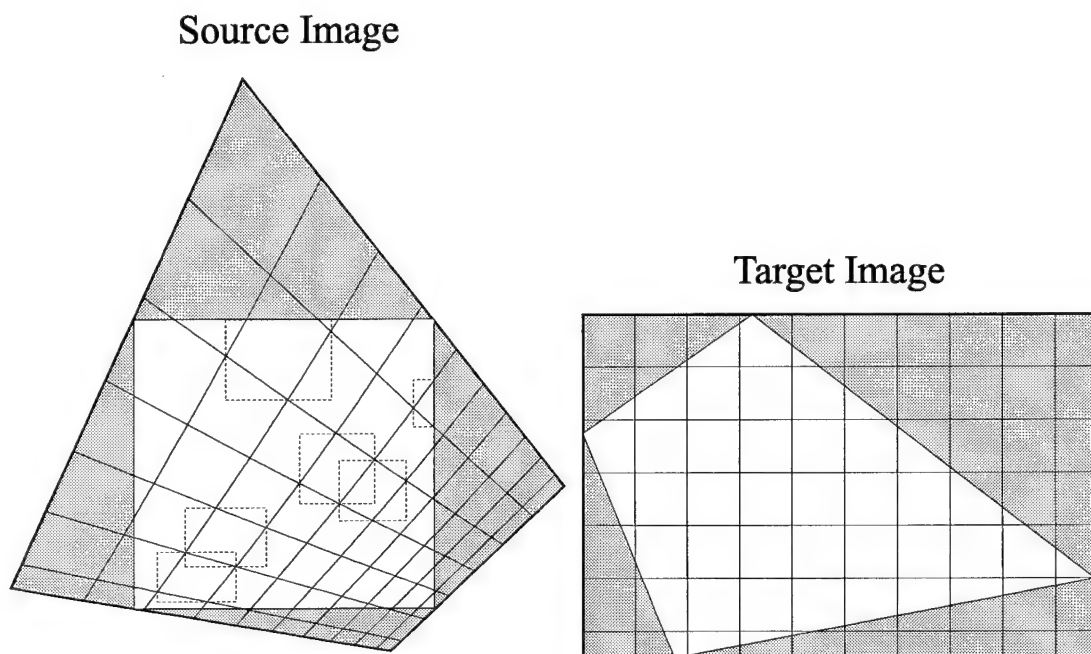
Figure 6: Block based image warping.
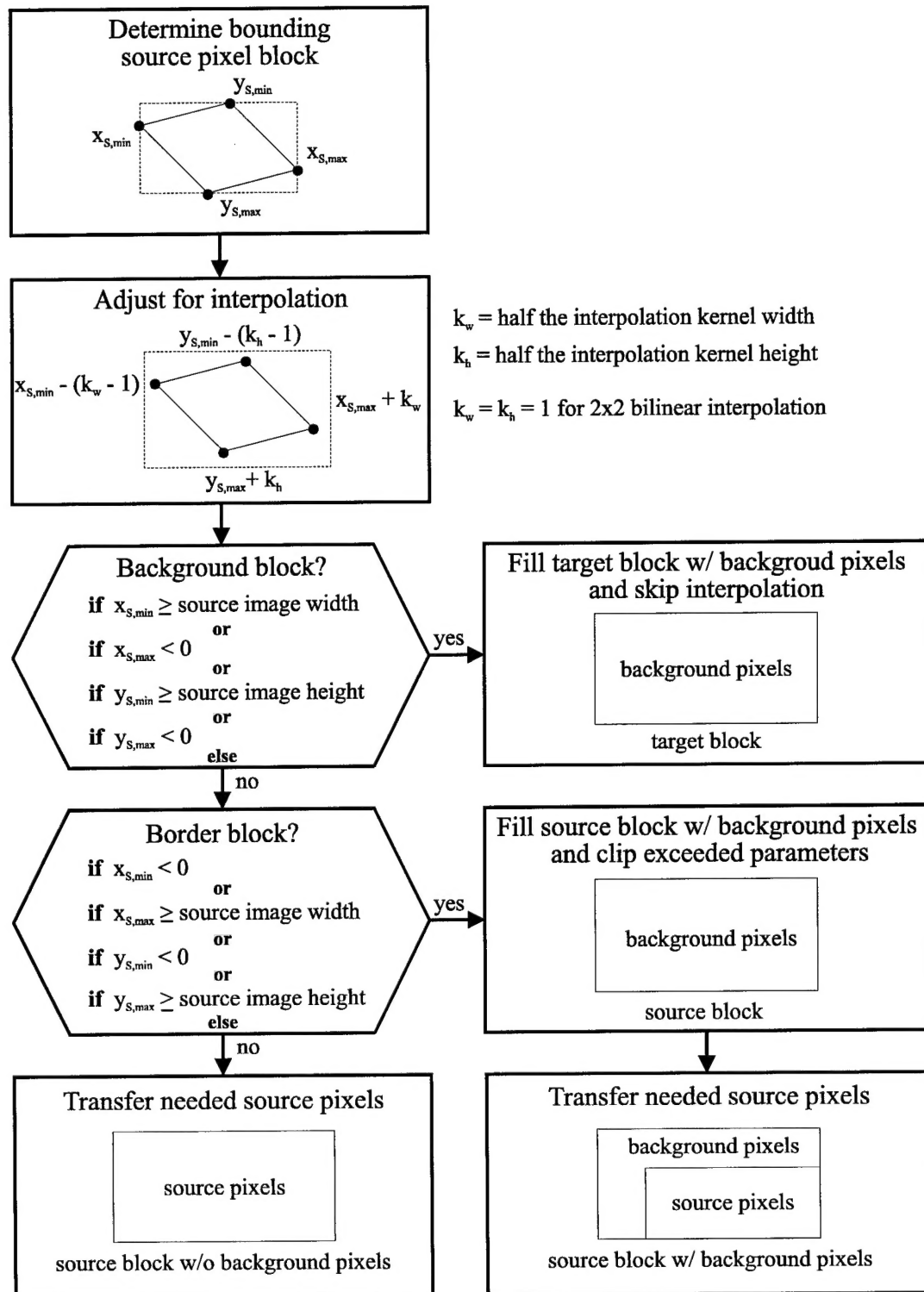
Figure 7: Source pixel block determination.

Source Image   Target Image
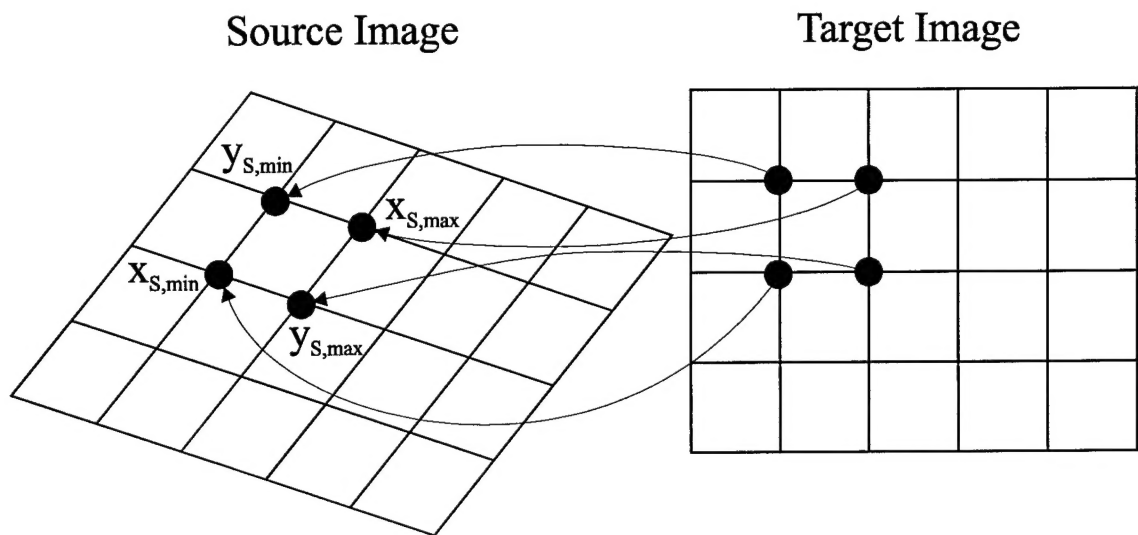
Figure 8: Affine warp bounding block.

Source Image   Target Image
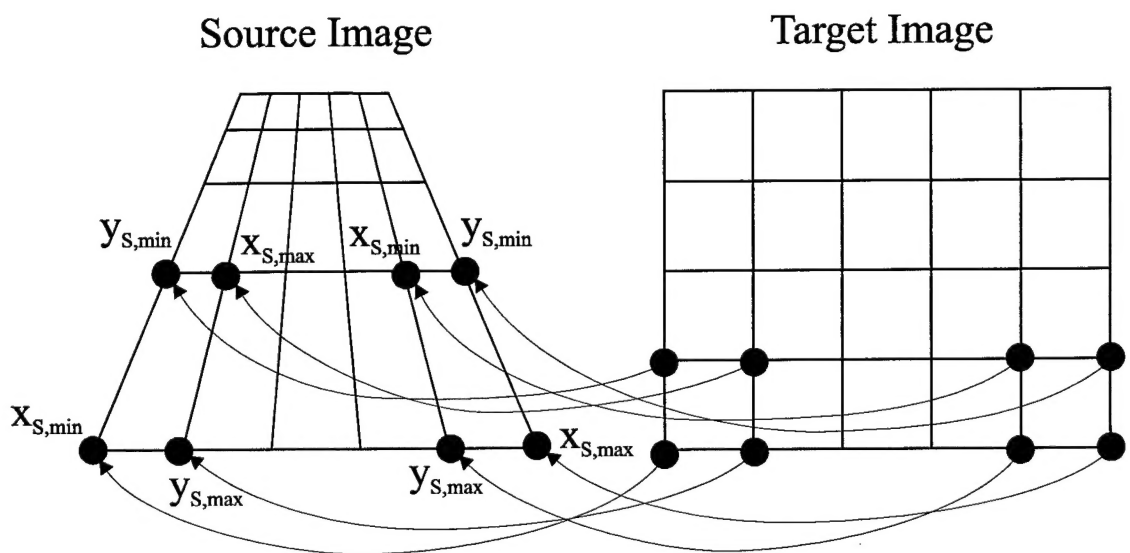
Figure 9: Perspective warp bounding block.
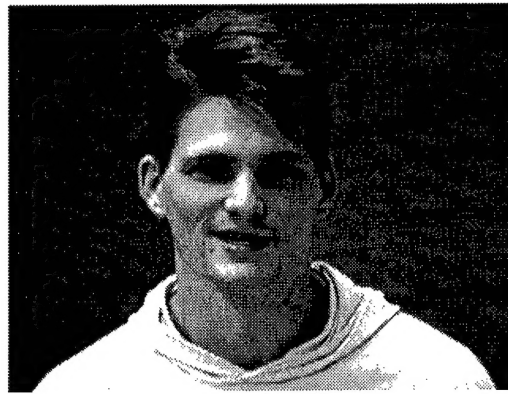
(a)　　　　　　　　　　　　(b)
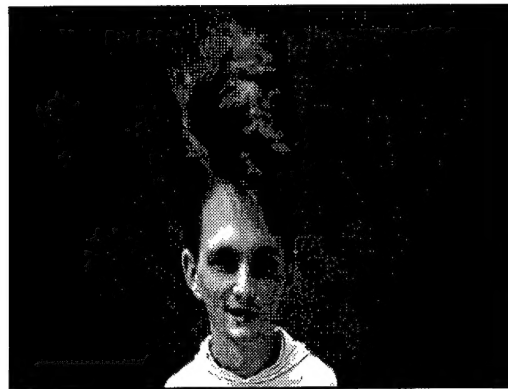

(c)　　　　　　　　　　　　(d)

Figure 10: Example affine warps. (a) original (b) (c) and (d) affine warps.

(a)

(b)

(c)

Figure 11: Example perspective warps. (a) original (b) and (c) perspective warps.